Chase Strickler

CSC 320-B

16 February 2018

Performance Analysis of Heap Sort

Heap sort is a quick, comparison based sorting algorithm that operates in place and makes use of recursion. My implementation of the algorithm constructed a binary tree as the heap within an array, where the left child of a given node, *i*, is $(2 * i) + 1$ and the right child is $(2 * i) + 2$. This makes it easy to navigate around the tree and find data as needed. The algorithm works on the precondition that each node is a max heap, or that the children of any given node are less than, or equal to, the parent. With that being said, the first step of heap sort is to build the max heap on the entire array, thus making the first value of the array, the largest. The algorithm then exchanges the root of the tree with the last element in the array, and repeats this entire process on a sub-array from 0 to *n* - 1, where *n* is the size of the array. This goes on until the sub-array is of size one. The expected speed of this algorithm is $\Theta(n(lgn))$, and I set up an experiment to test this claim.

I began my experiment by first implementing the heap sort algorithm, and testing its correctness on some arrays of selected values. After verifying it worked, I began adding counters in each of the methods wherever there was a swap or comparison. I then added the code to randomly generate ten arrays of equal size before increasing the size by a power of ten, from arrays of size ten to 100,000 inclusive. The results are recorded in the table below.

Size of Array	Average Inversions	Expected → $\Theta(n(lgn))$	Average / Expected
10	34	33	1.030
100	981	664	1.477
1,000	17,110	9,965	1.717
10,000	245,624	132,877	1.849
100,000	3,203,538	1,660,964	1.929

I included the expected results along with the quotient of the average and expected results. My hope in doing this was to see if there was some definite constant overhead on this algorithm, however it wasn't as obviously apparent as it was with insertion sort. With that being said, both the average and expected appear to be increasing logarithmically. I tried graphing the actual and expected results, but soon realized that both functions would appear to be *n*, as *lgn* is going to be a very small result and will not have a notable effect visually.

In conclusion, heap sort does in fact have a run time of $\Theta(n(lgn))$ in every case. This can be seen by comparing the actual run time to the expected run time, which both perform in $\Theta(n(lgn))$ time. There does exist some constant in the algorithm, however it isn't clear what that constant might be. Nonetheless, the constant doesn't make a notable difference, as heap sort still performs quite well while being spatially efficient.